Deployment & Orchestration with Terraform

Find the latest, print-friendly version of this presentation at https://christopherdemarco.com/terraform

The opinions and mistakes that follow are my own and do not represent my employer, Hashicorp, USENIX, or anyone else.

All code samples were believed correct at runtime.
Your mileage may vary.

To my grandfather, who taught me how to write.
To my father, who taught me why.

Who has used Terraform before?

Infrastructure Orchestration

** Not configuration management!*

All your vendors' web interfaces are different, and they all suck.

How do you document mouse-and-keyboard? Playback?!

How can creating infrastructure become repeatable and reliable?

Infrastructure as Code

Because programming makes things easier!

Automation!

Scale / test / rollback

Version

Collaborate / audit

Have you tried turning it off & back on again?

Interface consistency

Make devs help!

“Infrastructure” is broad:

- AWS • Google Compute Engine
- Azure • Oracle Public Cloud
- 1&1 • Digital Ocean • Scaleway
- VMWare • Docker • Heroku
- OpenStack • Kubernetes
- Rancher • Nomad
- CloudFlare • Dyn • DDNS
- Chef • Cobbler • Rundeck
- MySQL • PostgreSQL • RabbitMQ
- DataDog • Grafana • New Relic
- Icinga2 • Librato • StatusCake
- Mailgun • OpsGenie • PagerDuty

Stop making tools

I don’t want a drill bit,
I want a hole!

Stateful

Intelligent dependency graphing

Lightweight DSL

Incremental

Golang makes it fast and portable.

Commercial support available

Stop talking

show us something already

```
# hello.tf
provider "aws" { region = "us-west-1" }

resource "aws_instance" "hello_world" {
  ami = "ami-66eec506" # generic redhat image
  instance_type = "t2.micro"
  tags {
    Name = "Alice"
  }
}
```

Providers connect to a service that you'll be managing.

Resources are the things you want to manage. They are declared with a type and an identifier.

A resource has arguments. They can be strings, lists, or maps.

Whitespace is flexible.

HCL can be converted to/from JSON.

```
% alias tf=terraform
```

tf init

Initialize the package directory.
Copy in needed binaries.

Initialize empty state if none exists.

`tf init` is idempotent.

Terraform will prompt you to run `terraform init` if it can’t find what it needs.

```
% tf init

Initializing provider plugins...

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.aws: version = "~> 0.1"

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

tf plan

Interrogate the provider[s].

Compare with local state.

Calculate dependency graph.

Print what will be done.

The plan is not saved (unless expressly requested).

```
% tf plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.


------------------------------------------------------------------------

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + aws_instance.hello_world
      id:                               <computed>
      ami:                              "ami-66eec506"
      associate_public_ip_address:      <computed>
      availability_zone:                <computed>
      ebs_block_device.#:               <computed>
      ephemeral_block_device.#:         <computed>
      instance_state:                   <computed>
      instance_type:                    "t2.micro"
      ipv6_address_count:               <computed>
      ipv6_addresses.#:                 <computed>
      key_name:                         <computed>
      network_interface.#:              <computed>
      network_interface_id:             <computed>
      placement_group:                  <computed>
      primary_network_interface_id:     <computed>
      private_dns:                      <computed>
      private_ip:                       <computed>
      public_dns:                       <computed>
      public_ip:                        <computed>
      root_block_device.#:              <computed>
      security_groups.#:                <computed>
      source_dest_check:                "true"
      subnet_id:                        <computed>
      tags.%:                           "1"
      tags.Name:                        "Alice"
      tenancy:                          <computed>
      volume_tags.%:                    <computed>
      vpc_security_group_ids.#:         <computed>


Plan: 1 to add, 0 to change, 0 to destroy.

------------------------------------------------------------------------

Note: You didn't specify an "-out" parameter to save this plan, so Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.
```

tf apply

Plan.

Apply the dependency graph.

Update local state.

```
% tf apply
aws_instance.hello_world: Creating...
  ami:                               "" => "ami-66eec506"
  associate_public_ip_address:       "" => "<computed>"
  availability_zone:                 "" => "<computed>"
  ebs_block_device.#:                "" => "<computed>"
  ephemeral_block_device.#:          "" => "<computed>"
  instance_state:                    "" => "<computed>"
  instance_type:                     "" => "t2.micro"
  ipv6_address_count:                "" => "<computed>"
  ipv6_addresses.#:                  "" => "<computed>"
  key_name:                          "" => "<computed>"
  network_interface.#:               "" => "<computed>"
  network_interface_id:              "" => "<computed>"
  placement_group:                   "" => "<computed>"
  primary_network_interface_id:      "" => "<computed>"
  private_dns:                       "" => "<computed>"
  private_ip:                        "" => "<computed>"
  public_dns:                        "" => "<computed>"
  public_ip:                         "" => "<computed>"
  root_block_device.#:               "" => "<computed>"
  security_groups.#:                 "" => "<computed>"
  source_dest_check:                 "" => "true"
  subnet_id:                         "" => "<computed>"
  tags.%:                            "" => "1"
  tags.Name:                         "" => "Alice"
  tenancy:                           "" => "<computed>"
  volume_tags.%:                     "" => "<computed>"
  vpc_security_group_ids.#:          "" => "<computed>"
aws_instance.hello_world: Still creating... (10s elapsed)
aws_instance.hello_world: Still creating... (20s elapsed)
aws_instance.hello_world: Creation complete after 21s (ID: i-070957a0a7b61f1aa)

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

```
% tf apply
aws_instance.hello_world: Refreshing state... (ID: i-070957a0a7b61f1aa)

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```

Let’s change something.

```
# hello.tf
provider "aws" { region = "us-west-1" }

resource "aws_instance" "hello_world" {
  ami = "ami-66eec506" # generic redhat image
  instance_type = "t2.micro"
  tags {
    Name = "Bob"
  }
}
```

```
% tf plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

aws_instance.hello_world: Refreshing state... (ID: i-070957a0a7b61f1aa)

------------------------------------------------------------------------

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:

  ~ aws_instance.hello_world
      tags.Name: "Alice" => "Bob"


Plan: 0 to add, 1 to change, 0 to destroy.

------------------------------------------------------------------------

Note: You didn't specify an "-out" parameter to save this plan, so Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.
```

```
% tf apply
aws_instance.hello_world: Refreshing state... (ID: i-070957a0a7b61f1aa)
aws_instance.hello_world: Modifying... (ID: i-070957a0a7b61f1aa)
  tags.Name: "Alice" => "Bob"
aws_instance.hello_world: Modifications complete after 1s (ID: i-070957a0a7b61f1aa)

Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
```

What if it’s a destructive change?

```
# hello.tf
provider "aws" { region = "us-west-1" }

resource "aws_instance" "hello_world" {
  ami = "ami-7f15271f" # generic Ubuntu image
  instance_type = "t2.micro"
  tags {
    Name = "Bob"
  }
}
```

```
% tf plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

aws_instance.hello_world: Refreshing state... (ID: i-010b583b2f2bb33a5)

------------------------------------------------------------------------

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

-/+ aws_instance.hello_world (new resource required)
      id:                           "i-010b583b2f2bb33a5" => <computed> (forces new resource)
      ami:                          "ami-66eec506" => "ami-7f15271f" (forces new resource)
      associate_public_ip_address:  "true" => <computed>
      availability_zone:            "us-west-1b" => <computed>
      ebs_block_device.#:           "0" => <computed>
      ephemeral_block_device.#:     "0" => <computed>
      instance_state:               "running" => <computed>
      instance_type:                "t2.micro" => "t2.micro"
      ipv6_address_count:           "" => <computed>
      ipv6_addresses.#:             "0" => <computed>
      key_name:                     "" => <computed>
      network_interface.#:          "0" => <computed>
      network_interface_id:         "eni-7a736a79" => <computed>
      placement_group:              "" => <computed>
      primary_network_interface_id: "eni-7a736a79" => <computed>
      private_dns:                  "ip-172-31-2-29.us-west-1.compute.internal" => <computed>
      private_ip:                   "172.31.2.29" => <computed>
      public_dns:                   "ec2-54-193-12-19.us-west-1.compute.amazonaws.com" => <computed>
      public_ip:                    "54.193.12.19" => <computed>
      root_block_device.#:          "1" => <computed>
      security_groups.#:            "0" => <computed>
      source_dest_check:            "true" => "true"
      subnet_id:                    "subnet-6fe2ec29" => <computed>
      tags.%:                       "1" => "1"
      tags.Name:                    "Bob" => "Bob"
      tenancy:                      "default" => <computed>
      volume_tags.%:                "0" => <computed>
      vpc_security_group_ids.#:     "1" => <computed>


Plan: 1 to add, 0 to change, 1 to destroy.

------------------------------------------------------------------------

Note: You didn't specify an "-out" parameter to save this plan, so Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.
```

```
% tf apply
aws_instance.hello_world: Refreshing state... (ID: i-010b583b2f2bb33a5)
aws_instance.hello_world: Destroying... (ID: i-010b583b2f2bb33a5)
aws_instance.hello_world: Still destroying... (ID: i-010b583b2f2bb33a5, 10s elapsed)
aws_instance.hello_world: Still destroying... (ID: i-010b583b2f2bb33a5, 20s elapsed)
aws_instance.hello_world: Still destroying... (ID: i-010b583b2f2bb33a5, 30s elapsed)
aws_instance.hello_world: Still destroying... (ID: i-010b583b2f2bb33a5, 40s elapsed)
aws_instance.hello_world: Destruction complete after 50s
aws_instance.hello_world: Creating...
  ami:                          "" => "ami-7f15271f"
  associate_public_ip_address:  "" => "<computed>"
  availability_zone:            "" => "<computed>"
  ebs_block_device.#:           "" => "<computed>"
  ephemeral_block_device.#:     "" => "<computed>"
  instance_state:               "" => "<computed>"
  instance_type:                "" => "t2.micro"
  ipv6_address_count:           "" => "<computed>"
  ipv6_addresses.#:             "" => "<computed>"
  key_name:                     "" => "<computed>"
  network_interface.#:          "" => "<computed>"
  network_interface_id:         "" => "<computed>"
  placement_group:              "" => "<computed>"
  primary_network_interface_id: "" => "<computed>"
  private_dns:                  "" => "<computed>"
  private_ip:                   "" => "<computed>"
  public_dns:                   "" => "<computed>"
  public_ip:                    "" => "<computed>"
  root_block_device.#:          "" => "<computed>"
  security_groups.#:            "" => "<computed>"
  source_dest_check:            "" => "true"
  subnet_id:                    "" => "<computed>"
  tags.%:                       "" => "1"
  tags.Name:                    "" => "Bob"
  tenancy:                      "" => "<computed>"
  volume_tags.%:                "" => "<computed>"
  vpc_security_group_ids.#:     "" => "<computed>"
aws_instance.hello_world: Still creating... (10s elapsed)
aws_instance.hello_world: Creation complete after 15s (ID: i-0e72057272c073a0e)

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
```

What if we remove the resource?

```
# hello.tf
provider "aws" { region = "us-west-1" }

# resource "aws_instance" "hello_world" {
#   ami = "ami-7f15271f" # generic Ubuntu image
#   instance_type = "t2.micro"
#   tags {
#     Name = "Bob"
#   }
# }
```

```
% tf plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

aws_instance.hello_world: Refreshing state... (ID: i-0e72057272c073a0e)

------------------------------------------------------------------------

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  - aws_instance.hello_world


Plan: 0 to add, 0 to change, 1 to destroy.

------------------------------------------------------------------------

Note: You didn't specify an "-out" parameter to save this plan, so Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.
```

```
% tf destroy -force
aws_instance.hello_world: Refreshing state... (ID: i-0e72057272c073a0e)
aws_instance.hello_world: Destroying... (ID: i-0e72057272c073a0e)
aws_instance.hello_world: Still destroying... (ID: i-0e72057272c073a0e, 10s elapsed)
aws_instance.hello_world: Still destroying... (ID: i-0e72057272c073a0e, 20s elapsed)
aws_instance.hello_world: Still destroying... (ID: i-0e72057272c073a0e, 30s elapsed)
aws_instance.hello_world: Destruction complete after 40s

Destroy complete! Resources: 1 destroyed.
```

tf destroy

Teardown *all* resources.

Prompt for confirmation (unless `-force`).

There is no undo!

Calculate dependency graph so deletion is never* blocked.

```
% tf destroy -force
aws_instance.hello_world: Refreshing state... (ID: i-0f77f2f304fa398d7)
aws_instance.hello_world: Destroying... (ID: i-0f77f2f304fa398d7)
aws_instance.hello_world: Still destroying... (ID: i-0f77f2f304fa398d7, 10s elapsed)
aws_instance.hello_world: Still destroying... (ID: i-0f77f2f304fa398d7, 20s elapsed)
aws_instance.hello_world: Still destroying... (ID: i-0f77f2f304fa398d7, 30s elapsed)
aws_instance.hello_world: Still destroying... (ID: i-0f77f2f304fa398d7, 40s elapsed)
aws_instance.hello_world: Destruction complete after 50s

Destroy complete! Resources: 1 destroyed.
```

In a demo, no-one can see your creds.

Use the provider's default setup? (e.g. `~/.aws`)

Use env vars?

Set expressly?

Read a file?

Let’s build a simple webserver.

```
% tree
.
├── aws.tf
├── dns.tf
├── securitygroup.tf
├── terraform.tfstate
├── terraform.tfstate.backup
└── webserver.tf

0 directories, 6 files
```

Terraform will load and evaluate everything in the current directory named like `*.tf` .

Configuration is declarative, order does not matter.

Dependencies will be discovered and graphed.

```
# webserver.tf
resource "aws_eip" "webserver" {
  vpc = true
  instance = "${aws_instance.webserver.id}"
}

resource "aws_instance" "webserver" {
  ami = "ami-66eec506" # my prebuilt ami
  instance_type = "t2.micro"

  root_block_device {
    volume_size = "100"
    volume_type = "gp2"
  }

  vpc_security_group_ids = [
    "${aws_security_group.webserver.id}"
  ]

  tags { Name = "webserver" }
}

output "public_ip" {
  value = "${aws_eip.webserver.public_ip}"
}
```

Dependency graphing means that order does not matter!

Once a resource is instantiated, it exports attributes. Dereference them like `${type.name.attribute}` .

Outputs will become very useful later on.

```
# securitygroup.tf
resource "aws_security_group" "webserver" {
  ingress {
    from_port = 80 to_port = 80 protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port = 0 to_port = 0 protocol = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  tags { Name = "Foo" }
}
```

```
# dns.tf
resource "aws_route53_record" "webserver" {
  zone_id = "RLAFK5ZI1DKQ1"
  name = "webserver.christopherdemarco.com"
  type = "A"
  ttl = 60
  records = ["${aws_eip.webserver.public_ip}"]
}
```

Dependencies can span files . . . this can become difficult to manage. Use common sense.

`name` attribute is *not* Terraform's identifier!

```
  ephemeral_block_device.#:                   "" => "<computed>"
  instance_state:                             "" => "<computed>"
  instance_type:                              "" => "t2.micro"
  ipv6_address_count:                         "" => "<computed>"
  ipv6_addresses.#:                           "" => "<computed>"
  key_name:                                   "" => "<computed>"
  network_interface.#:                        "" => "<computed>"
  network_interface_id:                       "" => "<computed>"
  placement_group:                            "" => "<computed>"
  primary_network_interface_id:               "" => "<computed>"
  private_dns:                                "" => "<computed>"
  private_ip:                                 "" => "<computed>"
  public_dns:                                 "" => "<computed>"
  public_ip:                                  "" => "<computed>"
  root_block_device.#:                        "" => "1"
  root_block_device.0.delete_on_termination:  "" => "true"
  root_block_device.0.iops:                   "" => "<computed>"
  root_block_device.0.volume_size:            "" => "100"
  root_block_device.0.volume_type:            "" => "gp2"
  security_groups.#:                          "" => "<computed>"
  source_dest_check:                          "" => "true"
  subnet_id:                                  "" => "<computed>"
  tags.%:                                     "" => "1"
  tags.Name:                                  "" => "webserver"
  tenancy:                                    "" => "<computed>"
  volume_tags.%:                              "" => "<computed>"
  vpc_security_group_ids.#:                   "" => "1"
  vpc_security_group_ids.3296603073:          "" => "sg-0a65306c"
aws_instance.webserver: Still creating... (10s elapsed)
aws_instance.webserver: Still creating... (20s elapsed)
aws_instance.webserver: Creation complete after 22s (ID: i-01426c7aeffc499e9)
aws_eip.webserver: Creating...
  allocation_id:     "" => "<computed>"
  association_id:    "" => "<computed>"
  domain:            "" => "<computed>"
  instance:          "" => "i-01426c7aeffc499e9"
  network_interface: "" => "<computed>"
  private_ip:        "" => "<computed>"
  public_ip:         "" => "<computed>"
  vpc:               "" => "true"
aws_eip.webserver: Creation complete after 1s (ID: eipalloc-a076859d)

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

Outputs:

public_ip = 13.57.129.129
```

tf output

```
% tf output
public_ip = 13.57.129.129
% ssh $(tf output public_ip)
```

Sensitive outputs

```
output "username" { value = "hunter" }
output "password" { value = "hunter" sensitive = true }
```

```
% tf apply

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:

password = <sensitive>
username = hunter
```

```
% tf output
password = hunter
username = hunter
```

Variables

```
variable "name" { type = "string" }          # explicit type
variable "region" { default = "us-west-1" }  # inferred type

variable "amis" {
  default = {
    us-east-1 = "ami-53cb501d"
    us-west-1 = "ami-7f15271f"
  }
}

resource "aws_instance" "webserver" {
  instance_type = "t2.micro"
  ami = "${var.amis[var.region]}"
  tags { Name = "${var.name}" }
}
```

Variables must be declared. Type may be declared, or inferred from the default. If neither type nor default is given, a string is assumed.

Use variables just like you use resource attributes.

Variables are scoped to the package in which they are declared.

```
# terraform.tfvars
name = "Bob"
```

```
% tf plan -var "am_image=ami-f0f331d9" -var-file=dev.tfvars
```

```
% TF_VAR_somedir=$HOME tf plan
```

```
% tf plan
var.name
  Enter a value:
```

If `terraform.tfvars` exists, it will be evaluated.

Specify tfvars files, or set variables directly, on the command line.

Or set them via environment variables.

Terraform will prompt for values for any variables that haven’t been set otherwise.

```
# terraform.tfvars
name = "Bob"
```

```
% tf plan -var "am_image=ami-f0f331d9" -var-file=dev.tfvars
```

```
% TF_VAR_somedir=$HOME tf plan
```

```
% tf plan
var.name
  Enter a value:
```

Precedence:

The last file or variable specified on the command line wins.

Otherwise, `terraform.tfvars` wins.

Otherwise, an env var wins.

Otherwise, the default is used.

Otherwise, you're prompted.

*Maps are merged!

Variable syntax summary

```
variable "foo" { default = "foo" }              # a string is just a string
variable "one" { default = 1 }                  # integers don't need quotes
variable "pi"  { default = 3.141592358789 }     # nor do floats

variable "numbers" { default = [1, 2, 3,] }     # trailing commas are OK

variable "words" {
  default = {
    foo = "foo"                                 # maps have no commas
    bar = "bar"
    baz = "baz" mumble = [ "xyzzy", "plugh" ]   # and ignore whitespace
  }
}
```

Modules

```
% tree
.
├── aws.tf
├── modules
│   └── server
│       └── server.tf
├── terraform.tfstate
├── terraform.tfstate.backup
├── terraform.tfvars
└── web.tf

2 directories, 6 files
```

Modules are basically functions.

Recall that variables are scoped to a package. Therefore you must explicitly pass data into and out of modules.

```
# server.tf
variable "name" {}
variable "ami" {}
variable "security_group_ids" { type = "list" }

resource "aws_instance" "server" {
  ami = "${var.ami}"
  instance_type = "t2.micro"
  vpc_security_group_ids = ["${var.security_group_ids}"]
  tags { Name = "${var.name}" }
}

resource "aws_eip" "server" {
  vpc = true
  instance = "${aws_instance.server.id}"
}

output "public_address" { value = "${aws_eip.server.public_ip}" }
output "instance_id" { value = "${aws_instance.server.id}" }
```

Declare the variables that will be passed in.

Output the variables that will be returned.

```
# web.tf
variable "frontend_name" { default = "web" }
variable "backend_name" { default = "api" }
variable "amis" { type = "map" default = {} }

module "frontend" {
  source  = "modules/server"
  name = "${var.frontend_name}"
  ami = "${var.amis["frontend"]}" # note nested quotes
  security_group_ids = ["${aws_security_group.frontend.id}"]
}

module "backend" {
  source = "modules/server"
  name = "${var.backend_name}"
  ami = "${var.amis["backend"]}"
  security_group_ids = ["${aws_security_group.backend.id}"]
}

resource "aws_security_group" "frontend" {
  name = "frontend"
  ingress {
    from_port = 80 to_port = 80 protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress { from_port = 0 to_port = 0 protocol = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_security_group" "backend" {
  name = "backend"
  ingress {
    from_port = 8888 to_port = 8888 protocol = "tcp"
    cidr_blocks = ["${module.frontend.public_address}/32"]
  }
  egress { from_port = 0 to_port = 0 protocol = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

output "frontend" { value = "module.frontend.public_address" }
output "backend"  { value = "module.backend.public_address"  }
```

When passing variables into a module, make sure the names match!

```
% tf plan
Failed to load root config module: Error loading modules: module frontend: not found, may need to be downloaded using 'terraform get'
```

tf get

Copy the necessary modules into the working directory.

Local paths can be relative or absolute.

Load remote modules from Git, Mercurial, HTTP URIs; S3; Terraform Registry.

Read and evaluate any third-party modules before using them!

```
module "backup_me" {
  source = "../0937_modules/modules/server"
  name = "example"
  ami = "ami-327f532"
  security_group_ids = ["${aws_security_group.sg.id}"]
}

module "lambda_ami_backup" {
  source = "cloudposse/ec2-ami-backup/aws"
  name = "backup_me"
  stage = "dev"
  namespace = "backup_me"
  region = "us-west-1"
  ami_owner = "${var.account_id}"
  instance_id = "${module.backup_me.instance_id}"
  retention_days = "1"
  backup_schedule = "rate(5 minutes)"
}

% tf get
Get: file:///Users/demarco/cmd/terraform_class/0937_modules/modules/server
Get: https://api.github.com/repos/cloudposse/terraform-aws-ec2-ami-backup/tarball/0.2.3?archive=tar.gz
Get: git::https://github.com/cloudposse/tf_label.git?ref=tags/0.1.0
Get: git::https://github.com/cloudposse/tf_label.git?ref=tags/0.1.0
Get: git::https://github.com/cloudposse/tf_label.git?ref=tags/0.1.0
Get: git::https://github.com/cloudposse/tf_label.git?ref=tags/0.1.0
```

Provisioners

```
# provision_me.tf

variable "ssh_key_path" {
  default = {
    pub  = "~/.ssh/lisa.pub"
    priv = "~/.ssh/lisa"
  }
}

resource "aws_key_pair" "lisa" {
  key_name = "lisa"
  public_key = "${file(var.ssh_key_path["pub"])}"
}

resource "aws_instance" "provision_me" {
  instance_type = "t2.micro"
  ami = "ami-039ab163" # ubuntu
  key_name = "${aws_key_pair.lisa.key_name}"
  associate_public_ip_address = true
  vpc_security_group_ids = ["${aws_security_group.example.id}"]

  provisioner "remote-exec" {
    inline = [
      "until [ -f /var/lib/cloud/instance/boot-finished ]; do sleep 1; done",
      "sudo apt-get update",
      "sudo apt-get install -y python-minimal"
    ]
    connection {
      user = "ubuntu" # gotcha!
      private_key = "${file(var.ssh_key_path["priv"])}"
    }
  }
}
```

Provisioners get resources ready for the next step.

Maybe we use Ansible and need to ensure we have Python.

The provisioner will eventually timeout if, for example, you can’t SSH there from here.

Idempotent?

Multiple provisioners will be executed in the order they're declared.

Terraform does not "grok" provisioners. A failed provisioner run will cause the entire resource to fail.

OTOH: a successful provisioner will never be re-run. What if you *want* to re-run a provisioner?

tf taint

`tf taint` marks a resource as “tainted”—it is to be destroyed & recreated.

`tf untaint`

```
% tf plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

aws_key_pair.lisa: Refreshing state... (ID: lisa)
aws_security_group.example: Refreshing state... (ID: sg-196f3a7f)
aws_instance.provision_me: Refreshing state... (ID: i-01620df619a507e58)

------------------------------------------------------------------------

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

-/+ aws_instance.provision_me (tainted) (new resource required)
      id:                               "i-01620df619a507e58" => <computed> (forces new resource)
      ami:                              "ami-039ab163" => "ami-039ab163"
      associate_public_ip_address:      "true" => "true"
      availability_zone:                "us-west-1b" => <computed>
      ebs_block_device.#:               "0" => <computed>
      ephemeral_block_device.#:         "0" => <computed>
      instance_state:                   "running" => <computed>
      instance_type:                    "t2.micro" => "t2.micro"
      ipv6_address_count:               "" => <computed>
      ipv6_addresses.#:                 "0" => <computed>
      key_name:                         "lisa" => "lisa"
      network_interface.#:              "0" => <computed>
      network_interface_id:             "eni-ab1c05a8" => <computed>
      placement_group:                  "" => <computed>
      primary_network_interface_id:     "eni-ab1c05a8" => <computed>
      private_dns:                      "ip-172-31-13-159.us-west-1.compute.internal" => <computed>
      private_ip:                       "172.31.13.159" => <computed>
      public_dns:                       "ec2-52-53-154-45.us-west-1.compute.amazonaws.com" => <computed>
      public_ip:                        "52.53.154.45" => <computed>
      root_block_device.#:              "1" => <computed>
      security_groups.#:                "0" => <computed>
      source_dest_check:                "true" => "true"
      subnet_id:                        "subnet-6fe2ec29" => <computed>
      tenancy:                          "default" => <computed>
      volume_tags.%:                    "0" => <computed>
      vpc_security_group_ids.#:         "1" => "1"
      vpc_security_group_ids.1666365661: "sg-196f3a7f" => "sg-196f3a7f"


Plan: 1 to add, 0 to change, 1 to destroy.

------------------------------------------------------------------------

Note: You didn't specify an "-out" parameter to save this plan, so Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.
```

Is there a better way?

```
# provision_me.tf

variable "ssh_key_path" {
  default = {
    pub  = "~/.ssh/lisa.pub"
    priv = "~/.ssh/lisa"
  }
}

resource "aws_key_pair" "lisa" {
  key_name = "lisa"
  public_key = "${file(var.ssh_key_path["pub"])}"
}

resource "aws_instance" "example" {
  instance_type = "t2.micro"
  ami = "ami-039ab163" # ubuntu
  key_name = "${aws_key_pair.lisa.key_name}"
  associate_public_ip_address = true
  vpc_security_group_ids = ["${aws_security_group.example.id}"]
}

resource "null_resource" "provision_me" {
  triggers {
    instance_id = "${aws_instance.example.id}"
  }
  provisioner "remote-exec" {
    inline = [
      "until [ -f /var/lib/cloud/instance/boot-finished ]; do sleep 1; done",
      "sudo apt-get update",
      "sudo apt-get install -y python-minimal"
    ]
    connection {
      user = "ubuntu" # gotcha!
      private_key = "${file(var.ssh_key_path["priv"])}"
      host = "${aws_instance.example.public_ip}"
    }

  }
}
```

A `null_resource` is a “virtual resource” created in the project’s state.

To re-run the provisioner, `tf taint null_resource.provision_me`!

Other ways to use remote-exec

`inline` takes a list of strings.

`script` will upload a file and execute it.

`scripts` will upload a directory.

More provisioners

** Not configuration management!*

Chef

File

Local-exec

Salt-masterless

What if I want *n* of a thing? How do I loop?

```
variable "envs" { default = [ "dev", "qa", "prod" ] }

variable "count" { default = 3 }

resource "aws_instance" "web" {
  ami = "ami-039ab163"
  instance_type = "t2.micro"
  count = "${var.count}"
  tags { Name = "web${count.index}" Env = "${var.envs[count.index]}" }
}

resource "aws_eip" "web" {
  count = "${var.count}"
  instance = "${element(aws_instance.web.*.id, count.index)}"
}

output "web_ip" { value = "${aws_eip.web.*.public_ip}" }
```

Terraform iterates over `count`, setting `count.index` each time.

If `count` is > 3, we will get duplicate Env tags.

`*` means “all resources”.

Select into a list with the `element()` function.

OK, how about conditionals?

This almost works . . .

```
module "frontend" {
  source = "modules/web"
  is_public = 1
}
module "backend" {
  source = "modules/web"
  is_public = 0
}
```

```
# modules/web/web.tf

variable "is_public" { }
resource "aws_instance" "web" {
  ami = "ami-039ab163"
  instance_type = "t2.micro"
}
resource "aws_eip" "web" {
  count = "${var.is_public}"
  instance = "${aws_instance.web.id}"
}
```

But the ternary operator is more flexible.

```
# modules/web/web.tf

variable "is_public" { }

resource "aws_instance" "web" {
  ami = "ami-039ab163"
  instance_type = "t2.micro"
  vpc_security_group_ids = ["${aws_security_group.web.id}"]
}
resource "aws_eip" "web" {
  count = "${var.is_public ? 1 : 0}"
  instance = "${aws_instance.web.id}"
}
resource "aws_security_group" "web" {
  name = "web"
  ingress {
    from_port = 80 to_port = 80 protocol = "tcp"
    cidr_blocks = [
      "${var.is_public ? "0.0.0.0/0" : "127.0.0.1/32"}"
    ]
  }
  egress {
    from_port = 0 to_port = 0 protocol = -1
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

Use functions to modify variables.

https://www.terraform.io/docs/configuration/interpolation.html

tf console

```
% tf console
> "${format("host-%03d", 23)}"
host-023
```

```
% echo '"${urlencode(title("hello world"))}"' |
tf console
Hello+World
```

```
# terraform.tf
variable "passwd" { }
```

```
% tf console -var "passwd=$(grep $(whoami) /etc/passwd)"
> "${slice(split(":", var.passwd), 6, 7)}"
[
  /usr/bin/zsh
]
```

Templates

Use templates where string interpolation would be unwieldy.

```
variable "name" { default = "world" }

data "template_file" "hello" {
  template = "Hello $${name}!"
  vars {
    name = "${var.name}"
  }
}

resource "null_resource" "hello" {
  provisioner "local-exec" {
    command = "echo ${data.template_file.hello.rendered}"
  }
}
```

Data sources are dynamic, read-only ways to get at data.

Escape interpolation with `$$`.

Like modules, variables must be passed into a template expressly.

Unlike modules, template variables don’t get declared.

Also unlike modules, template variables are not accessed like `var.name`.

```
% tf apply
data.template_file.hello: Refreshing state...
null_resource.hello: Creating...
null_resource.hello: Provisioning with 'local-exec'...
null_resource.hello (local-exec): Executing: ["/bin/sh" "-c" "echo Hello world!"]
null_resource.hello (local-exec): Hello world!
null_resource.hello: Creation complete after 0s (ID: 6413292886236504909)

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
% tf apply
data.template_file.hello: Refreshing state...
null_resource.hello: Refreshing state... (ID: 6413292886236504909)

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```

Well, that was anticlimactic . . .

```
variable "clustername" { default = "megamaid" }
variable "username" { default = "skroob" }
variable "password" { default = "12345" }

resource "aws_instance" "ecs_host" {
  count = "${var.count}"
  name = "ecs_host_${count.index}"
  ami = "ami-09d2fb69"
  instance_type = "t2.micro"
  user_data = "${data.template_file.script.rendered}"
  tags {
    Name = "${ecs_host_${count.index}"
    ClusterName = "${var.clustername}"
}

data "template_file" "script" {
  template = "${file("ecs_init.sh.tmpl")}"
  vars {
    clustername = "${var.clustername}"
    username = "${var.username}"
    password = "${var.password}"
  }
}

resource "local_file" "debug" {
  content = "${data.template_file.script.rendered}"
  filename = "out"
}
```

AWS lets you pass “user data”: scripts that hosts will run on first boot.

Load a template from a file with the `file()` function.

Render the template locally to assist debugging.

```
#!/bin/bash
# ecs_init.sh.tmpl

cat <<'EOF' >> ecs.config
ECS_CLUSTER=${clustername}
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":{"username":"${username}","password":"${password},"email":"email@example.com"}}
ECS_LOGLEVEL=debug
EOF
```

One-to-many is easy; what if we want many-to-one?

```
% cat .tmuxinator/example.yml
name: example
windows:
  - hosts:
      layout: even-vertical
      panes:
        - ssh -q ubuntu@54.183.60.135
        - ssh -q ubuntu@54.67.98.39
        - ssh -q ubuntu@54.241.140.120
```

```
data "template_file" "tmuxinator_host" {
  count = "${var.count}"
  template = "        - ssh -q ubuntu@$${address}"
  vars {
    address = "${element(aws_instance.host.*.public_ip, count.index)}"
  }
}

data "template_file" "tmuxinator_wrapper" {
  template = <<EOF
name: example
windows:
  - hosts:
      layout: even-vertical
      panes:
$${hosts}
EOF
  vars = {
    hosts = "${join("\n", data.template_file.tmuxinator_host.*.rendered)}"
  }
}

resource "local_file" "tmuxinator" {
  content = "${data.template_file.tmuxinator_wrapper.rendered}"
  filename = "${var.dir}/.tmuxinator/example.yml"
}
```

Here-doc syntax is available throughout Terraform.

The `local_file` resource will create directories as needed.

```
variable "somelist" { default = ["foo", "bar", "baz"] }
variable "somestring" { default = "this secret password" }

data "template_file" "json" {
  template = <<EOF
{
"someparams": $${somelist},
"password": $${jsonencode(bcrypt(somestring, 10))}
}
EOF
  vars = {
    somelist = "${jsonencode(var.somelist)}"
    somestring = "${var.somestring}"
  }
}

resource "local_file" "output" {
  content = "${data.template_file.json.rendered}"
  filename = "somefile.json"
}
```

You can use functions in templates, too.

```
{
"someparams": ["foo","bar","baz"],
"password": "$2a$10$q4g/T9gDOXuqhhXIyZ1x2Ope5DaW4Lb94JCN6L.mZUrzPkFWATbpm"
}
```

break

return at 1100

What happens if I lose state?

```
% tf apply
aws_security_group.example: Creating...
  description: "" => "Managed by Terraform"
  egress.#:    "" => "<computed>"
  ingress.#:   "" => "<computed>"
  name:        "" => "example"
  owner_id:    "" => "<computed>"
  vpc_id:      "" => "<computed>"
Error applying plan:

1 error(s) occurred:

* aws_security_group.example: 1 error(s) occurred:

* aws_security_group.example: Error creating Security Group: InvalidGroup.Duplicate: The security group 'example' already exists for VPC 'vpc-ea7c8f8f'
        status code: 400, request id: 64fdbc4d-84a4-4d5c-a665-c80262b3ca3c

Terraform does not automatically rollback in the face of errors.
Instead, your Terraform state file has been partially updated with
any resources that successfully completed. Please address the error
above and apply again to incrementally change your infrastructure.
```

Use remote state!

```
% echo 'terraform.tfstate*' >> .gitignore
```

Backup

Collaboration

Pipe data among tf projects

Pipe data among other tools

Update centralized SSOT

State backends

Artifactory

AWS S3

Azure

Consul

etcd

Google Cloud Storage

HTTP REST

OpenStack

```
provider "aws" { region = "us-west-1" }

terraform {
  backend "s3" {
    bucket = "cmd-lisa-state"
    key = "terraform.tfstate"
    region = "us-west-1"
  }
}

resource "aws_security_group" "example" { name = "example" }
```

Backends have specific configuration details.

Whichever backend you use, ensure that it is not publicly visible.

Turn on versioning if available.

```
% tf init

Initializing the backend...
Do you want to copy state from "local" to "s3"?
  Pre-existing state was found in "local" while migrating to "s3". No existing
  state was found in "s3". Do you want to copy the state from "local" to
  "s3"? Enter "yes" to copy and "no" to start with an empty state.

  Enter a value: yes


Successfully configured the backend "s3"! Terraform will automatically
use this backend unless the backend configuration changes.

Initializing provider plugins...

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.aws: version = "~> 0.1"

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

If the backend is already configured, `tf init` will proceed.

Existing state will be migrated to the new backend.

```
provider "aws" { region = "us-west-1" }

terraform {
  backend "s3" {
    # bucket = "cmd-lisa-state"
    # key = "terraform.tfstate"
    # region = "us-west-1"
  }
}

resource "aws_security_group" "example" { name = "example" }
```

```
% tf init \
> -backend-config="bucket=cmd-lisa-state" \
> -backend-config="key=somename/terraform.tfstate" \
> -backend-config="region=us-west-1"
```

If the backend is not configured, use `tf init` to fill in the details.

This is useful for automated configuration.

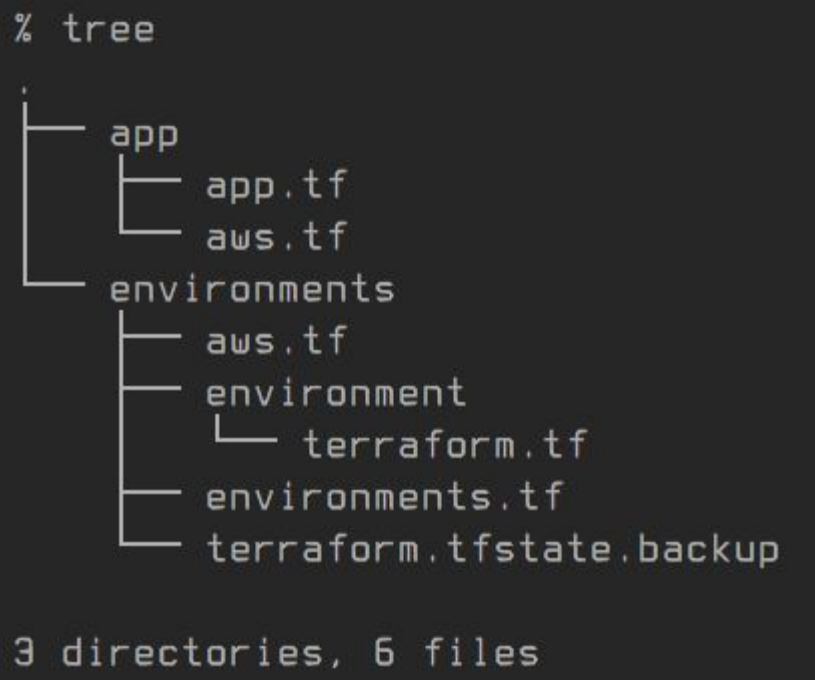
```
#!/bin/bash

cd /path/to/terraform/package
for ip in $(tf output); do
    curl -s http://$ip/alive >/dev/null || echo DOWN
done
```

Now that state is stored remotely, we can easily use it in other parts of our infrastructure.

`tf state pull` will fetch remote state.

Reference state outside of the package.

Security groups and an app



```
# environments.tf

terraform {
  backend "s3" {
    bucket = "cmd-lisa-example"
    key = "environments/terraform.tfstate"
    region = "us-west-1"
  }
}

module "qa" {
  source = "environment"
  name = "qa"
}

module "prod" {
  source = "environment"
  name = "prod"
}

output "qa" { value = "${module.qa.id}" }
output "prod" { value = "${module.prod.id}" }


# app.tf

data "terraform_remote_state" "environments" {
  backend = "s3"
  config {
    bucket = "cmd-lisa-example"
    key = "environments/terraform.tfstate"
    region = "us-west-1"
  }
}

resource "aws_instance" "app" {
  ami = "ami-66eec506"
  instance_type = "t2.micro"
  vpc_security_group_ids = ["${data.terraform_remote_state.environments.qa}"]
}
```

Reference remote state just like any other resource.

If the remote state changes, any references will become stale.

Re-apply any referring packages.

```
% tf destroy -force
data.terraform_remote_state.environments: Refreshing state...
aws_instance.app: Refreshing state... (ID: i-0ba24e155a75dc3d5)
aws_instance.app: Destroying... (ID: i-0ba24e155a75dc3d5)
aws_instance.app: Still destroying... (ID: i-0ba24e155a75dc3d5, 10s elapsed)
aws_instance.app: Still destroying... (ID: i-0ba24e155a75dc3d5, 20s elapsed)
aws_instance.app: Still destroying... (ID: i-0ba24e155a75dc3d5, 30s elapsed)
aws_instance.app: Destruction complete after 31s

Destroy complete! Resources: 1 destroyed.
```

```
% cd ../environments
% tf plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

aws_security_group.environment: Refreshing state... (ID: sg-15c04973)
aws_security_group.environment: Refreshing state... (ID: sg-dbce47bd)

------------------------------------------------------------------------

No changes. Infrastructure is up-to-date.

This means that Terraform did not detect any differences between your
configuration and real physical resources that exist. As a result, no
actions need to be performed.
```

Happily, `terraform_remote_state` is a read-only reference. `destroy` does not touch it.

Inspect state.

tf show

Dump the state or planfile.

This is useful for figuring out how to reference a module we want to taint.

(But we can do better!)

```
% tf show | head -30
aws_eip.webserver:
  id = eipalloc-c451a2f9
  association_id = eipassoc-0ad78036
  domain = vpc
  instance = i-0506975b33c87158b
  network_interface = eni-b37f67b0
  private_ip = 172.31.14.52
  public_ip = 13.56.4.11
  vpc = true
aws_instance.webserver:
  id = i-0506975b33c87158b
  ami = ami-66eec506
  associate_public_ip_address = true
  availability_zone = us-west-1b
  disable_api_termination = false
  ebs_block_device.# = 0
  ebs_optimized = false
  ephemeral_block_device.# = 0
  iam_instance_profile =
  instance_state = running
  instance_type = t2.micro
  ipv6_addresses.# = 0
  key_name =
  monitoring = false
  network_interface.# = 0
  network_interface_id = eni-b37f67b0
  primary_network_interface_id = eni-b37f67b0
  private_dns = ip-172-31-14-52.us-west-1.compute.internal
  private_ip = 172.31.14.52
  public_dns = ec2-13-57-3-112.us-west-1.compute.amazonaws.com
```

tf state

Select and pretty-print items from the state.

```
% tf state list
aws_eip.webserver
aws_instance.webserver
aws_security_group.webserver
```

tf state pull | jq

`pull` will dump the state as JSON.

Now you can use the AWESOME `jq` JSON parser!

Use workspaces to namespace state.

tf workspace

Use distinct directories for state.

`new` to create, `list` to list, `select` to switch. `show` if you can’t remember where you are.

Use with the Consul and S3 backends.

Use the variable `${terraform.workspace}` for e.g. naming things.

Enough AWS already!
What else can Terraform orchestrate?

Let’s take a tour.
Beware the bright light!

Google Cloud

```
provider "google" {
  credentials = "${file("keys/project.json")}"
  project = "project-941302"
  region = "us-east4"
}

resource "google_compute_instance" "foo" {
  name = "foo"
  machine_type = "n1-standard-1"
  zone = "us-east4a"
  boot_disk {
    initialize_params { image = "ubuntu-1704" }
  }
  network_interface {
    network = "default"
    access_config { }
  }
  metadata {
    sshKeys = "someuser:${file("ssh_pubkey")}"
  }
}

data "google_dns_managed_zone" "foo" {
  name = "christopherdemarco.com"
}

resource "google_dns_record_set" "foo" {
  name = "${google_compute_instance.foo.name}.christopherdemarco.com"
  type = "A"
  ttl = 60
  managed_zone = "${data.google_dns_managed_zone.foo.name}"
  rrdatas = [
    "${google_compute_instance.foo.network_interface.0.access_config.0.assigned_nat_ip}"
  ]
}
```

PostgreSQL

```
provider "aws" { region = "us-west-1" }

resource "aws_db_instance" "example" {
  name = "example"
  username = "skroob"
  password = "12345678"
  instance_class = "db.t2.small"
  allocated_storage = "10"
  storage_type = "standard"
  engine = "postgres"
  skip_final_snapshot = "true"
  publicly_accessible = "true"
  vpc_security_group_ids = ["${aws_security_group.example.id}"]
}

resource "aws_security_group" "example" {
  ingress {
    from_port = 5432 to_port = 5432 protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

provider "postgresql" {
  database = "postgres"
  host = "${aws_db_instance.example.address}"
  port = 5432
  username = "skroob"
  password = "12345678"
}

variable "username" { default = "hunter" }
variable "password" { default = "hunter" }

resource "postgresql_database" "db" {
  name = "mydb"
}

resource "postgresql_role" "user" {
  name = "${var.username}"
  login = true
  password = "${var.password}"
}

output "endpoint" { value = "${aws_db_instance.example.address}" }
```

And of course, you can mix-and-match.

```
provider "google" {
  credentials = "${file("key.json")}"
  project = "magrathea-178523"
  region = "us-east4"
}
provider "aws" { region = "us-west-1" }

resource "google_compute_instance" "example" {
  count = "2"
  name = "example${count.index}"
  machine_type = "f1-micro"
  zone = "us-east4-a"
  boot_disk { initialize_params { image = "ubuntu-1704" } }
  network_interface { network = "default" access_config { } }
}

data "google_dns_managed_zone" "example" { name = "foamninja" }

resource "google_dns_record_set" "example" {
  name = "example.${data.google_dns_managed_zone.example.dns_name}"
  type = "A"
  ttl = 60
  managed_zone = "${data.google_dns_managed_zone.example.name}"
  rrdatas = ["${google_compute_instance.example.*.network_interface.0.access_config.0.assigned_nat_ip}"]
}

resource "null_resource" "example" {
  triggers { dns_record = "${google_dns_record_set.example.id}" }
  provisioner "local-exec" { command = "sleep 60" }
}

data "dns_a_record_set" "example" {
  host = "example.foam.ninja"
  depends_on = ["null_resource.example"]
}

resource "aws_security_group" "example" {
  ingress {
    from_port = 80 to_port = 80 protocol = "tcp"
    cidr_blocks = ["${formatlist("%s/32", data.dns_a_record_set.example.addrs)}"]
  }
  egress { from_port = 0 to_port = 0 protocol = "-1" cidr_blocks = ["0.0.0.0/0"]
  }
}
```

Recap: providers, resources & data sources

Provider arguments

Providers vary as to authentication.

Creds may be provided inline, via environment variables, and/or filesystem paths.

Providers may require regions, cloud types, and/or projects to be specified.

Store your creds safely!

Resource arguments and attributes

Arguments to resources may be required or optional.

Some arguments can be changed without re-creating the resource.

An attribute’s value always reflects its current state. Refresh via `plan` or `apply`.

Data sources

Data sources are *read-only*.

Arguments provide filters to restrict the kind / quantity of results.

Attributes are typically numerous and contain nested data structure.

Gotchas

Gotcha!
Module param mismatch

```
# package
module "foo" {
  source = "module"
  magic = "xyzzy"
}
```

```
# module
variable "magic_word" { }
output "magic_word" { value = "${var.magic_word}" }
```

```
% tf apply
1 error(s) occurred:

* module root:
    module foo: magic is not a valid parameter
    module foo: required variable "magic_word" not set
```

Gotcha!
Silent failure on missing output

```
module "example" {
  source = "module"
}

resource "aws_security_group" "example" {
  ingress {
    from_port = 0 to_port = 0 protocol = "-1"
    cidr_blocks = ["${module.example.public_ip}/32"]
  }
}

output "public_ip" { value = "${module.example.public_ip}" }
```

```
# module

resource "aws_eip" "example" { }

output "public_ip" { value = "${aws_eip.example.public_io}" }
```

Silent failure on missing output (cont.)

```
% tf apply
module.example.aws_eip.example: Creating...
  allocation_id:      "" => "<computed>"
  association_id:     "" => "<computed>"
  domain:             "" => "<computed>"
  instance:           "" => "<computed>"
  network_interface:  "" => "<computed>"
  private_ip:         "" => "<computed>"
  public_ip:          "" => "<computed>"
  vpc:                "" => "<computed>"
module.example.aws_eip.example: Creation complete after 0s (ID: eipalloc-6e6c9f53)
aws_security_group.example: Creating...
  description:                                "" => "Managed by Terraform"
  egress.#:                                   "" => "<computed>"
  ingress.#:                                  "" => "1"
  ingress.~2281976595.cidr_blocks.#:          "" => "<computed>"
  ingress.~2281976595.description:            "" => ""
  ingress.~2281976595.from_port:              "" => "0"
  ingress.~2281976595.ipv6_cidr_blocks.#:     "" => "0"
  ingress.~2281976595.protocol:               "" => "-1"
  ingress.~2281976595.security_groups.#:      "" => "0"
  ingress.~2281976595.self:                   "" => "false"
  ingress.~2281976595.to_port:                "" => "0"
  name:                                       "" => "<computed>"
  owner_id:                                   "" => "<computed>"
  vpc_id:                                     "" => "<computed>"
aws_security_group.example: Creation complete after 1s (ID: sg-f8e0b59e)

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
% tf output
The state file either has no outputs defined, or all the defined
outputs are empty. Please define an output in your configuration
with the `output` keyword and run `terraform refresh` for it to
become available. If you are using interpolation, please verify
the interpolated value is not empty. You can use the
`terraform console` command to assist.
```

Gotcha! Useless var interpolation!

```
variable "words" { default = { foo = "FOO" bar = "BAR" } }

variable "myword" { default = "${var.words["foo"]}" }
```

```
% tf apply
1 error(s) occurred:

* module root: 1 error(s) occurred:

* Variable 'myword': cannot contain interpolations
```

Use `local` variables.

```
locals {

  "words" {
    foo = "FOO"
    bar = "BAR"
  }

  "myword" =  "${local.words["foo"]}"
}

output "myword" { value = "${local.myword}" }
```

Gotcha! Multiple accounts

```
provider "aws" {
  region = "us-west-1"
}

provider "aws" {
  region = "us-west-1"
  alias = "legacy"
  profile = "legacy"
}

resource "aws_instance" "foobar" {
  ami = "ami-3ab1fa2d"
  type = "t2.micro"
  associate_public_ip_address = true
}

resource "aws_route53_record" "example" {
  provider = "aws.legacy"
  zone_id  = "AK194BZ86FKQ1"
  name = "foobar.example"
  type = "A"
  ttl = 60
  records = ["${aws_instance.foobar.public_ip}"]
}
```

Gotcha!
Bizarre errors

```
* aws_instance.example: Error launching source instance: InvalidAMIID.NotFound: The image id '[ami-a4c7edb2]' does not exist
        status code: 400, request id: 40320df0-a100-4be4-bbf0-1022ebd0d274
```

Gotcha!
State changed from under me!

```
% tf plan -out=planfile

% tf apply planfile
```

Gotcha!
Destroy is scary!

Insurance against co-workers?

```
resource "aws_instance" "Super-Important" {
  instance_type = "t2.micro"
  ami = "ami-039ab163"

  lifecycle { prevent_destroy = true }
}
```

Gotcha!
Timeout!

```
resource "aws_instance" "provision_me" {
  instance_type = "t2.micro"
  ami = "ami-039ab163"
  key_name = "somekey"
  associate_public_ip_address = true
  vpc_security_group_ids = ["${aws_security_group.somegroup.id}"]

  timeouts {
    create = "20m"
    update = "1h"
    delete = "86400s"
  }

  provisioner "remote-exec" {
    connection {
      user = "ubuntu" # gotcha!
      private_key = "${file(var.ssh_key_path["priv"])}"
      timeout = "20m"
    }

    inline = [
      "until [ -f /var/lib/cloud/instance/boot-finished ]; do sleep 1; done",
      "sudo apt-get update",
      "sudo apt-get install -y python-minimal"
    ]
  }
}
```

Gotcha! Increase verbosity

`TF_LOG=<TRACE, DEBUG, WARN, INFO, ERROR>`

You get to drink from the firehose!

It’s mainly useful for proving you’ve found a bug.

Gotcha!
A bug in the provider!

Use the provider's `version` argument to pin it.

Providers are in a separate GitHub organization from Terraform Core: https://github.com/terraform-providers .
Look at the relevant CHANGELOG.

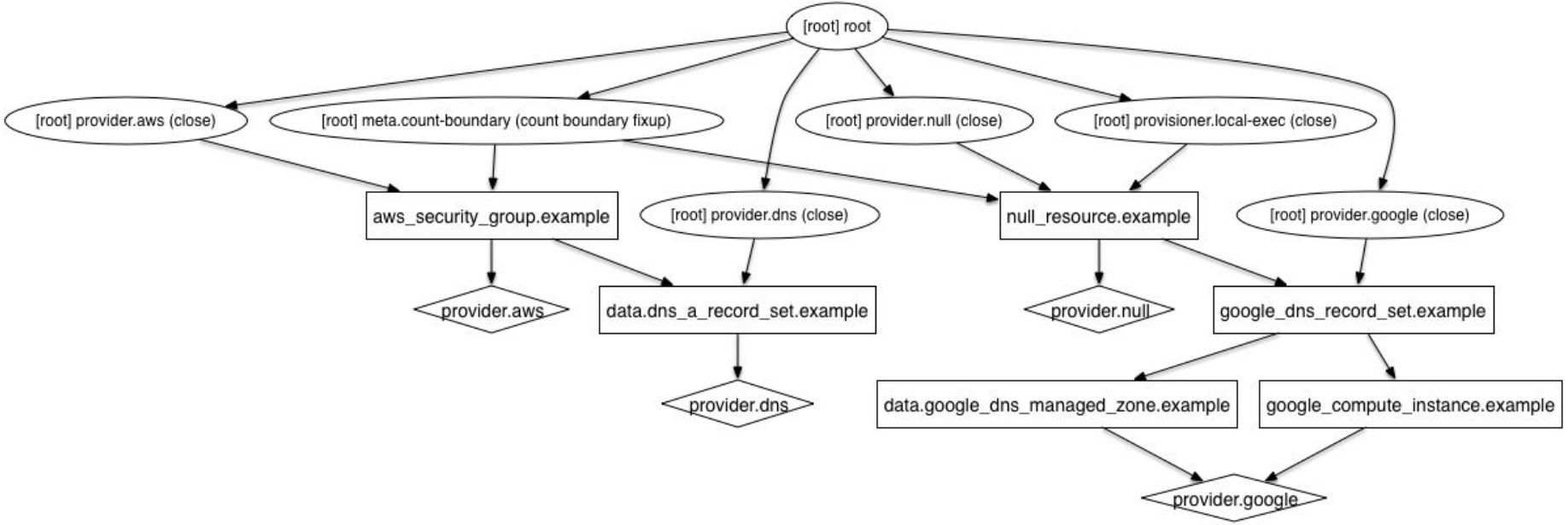
Gotcha!
tfstate is huge!

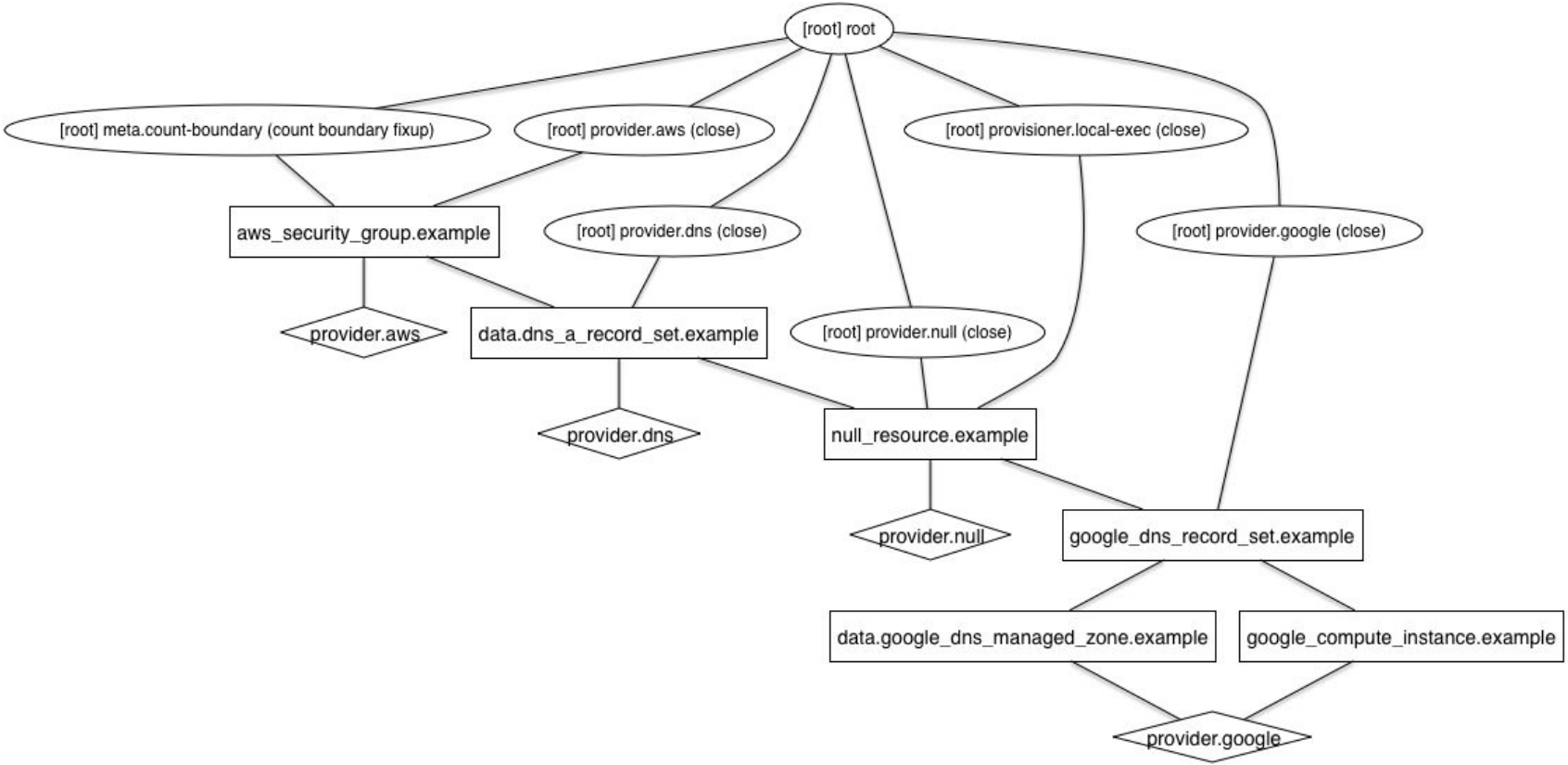
It’s just JSON. Use `jq` or your favorite programming language.

Go install `jq` and rejoice.

Gotcha! Dependencies hurt my brain!

```
% tf graph | dot -Tpng > terraform.png

% tf graph > terraform.graffle
```


[root] root
[root] provider.aws (close)
[root] meta.count-boundary (count boundary fixup)
[root] provider.null (close)
[root] provisioner.local-exec (close)
aws_security_group.example
[root] provider.dns (close)
null_resource.example
[root] provider.google (close)
provider.aws
data.dns_a_record_set.example
provider.null
google_dns_record_set.example
provider.dns
data.google_dns_managed_zone.example
google_compute_instance.example
provider.google


[root] root
[root] meta.count-boundary (count boundary fixup)
[root] provider.aws (close)
[root] provisioner.local-exec (close)
aws_security_group.example
[root] provider.dns (close)
[root] provider.google (close)
provider.aws
data.dns_a_record_set.example
[root] provider.null (close)
provider.dns
null_resource.example
provider.null
google_dns_record_set.example
data.google_dns_managed_zone.example
google_compute_instance.example
provider.google

. . . Getcha? Golang niceities

Terraform is a single binary. Download from http://terraform.io .

`tf fmt` will apply the standard Go formatter to canonicalize style—spacing, quoting, etc.

`tf validate` will check for syntax and dependency graph issues.

Q & A

Thank you.

Please fill out your surveys.